Applied Cloning Techniques for a Genetic Algorithm Used in Evolvable Hardware Design

Viet C. Trinh
vtrinh@isl.ucf.edu

Gregory A. Holifield
greg.holifield@us.army.mil

Annie S. Wu
aswu@cs.ucf.edu

School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816, USA

ABSTRACT

Genetic algorithms are commonly used to perform searches on complex search spaces for optimum solutions of many real-world problems. The evolvable hardware domain presents many problems with complex search spaces subject to the use of a genetic algorithm as an optimization technique for field programmable gate arrays (FPGA) implementations. The optimization of FPGA implementations, in general, are naturally difficult problems to solve using conventional genetic algorithm techniques due to the large number of local optima points. Therefore, an alternative cloning strategy is proposed which introduces a more powerful and diverse selection method taken directly from artificial biological principles. Performance studies of the new method provides good insight on the application of cloning to this domain.

Keywords: Genetic Algorithms, Evolutionary Computing, Applied Cloning, Evolvable Hardware, FPGA Design.

1. INTRODUCTION

Background

Evolvable hardware is capable of realizing optimized circuits beyond those of conventional design of logic circuits, in effect, the one-bit adder, two-bit adder, and two-bit multiplier described in the Miller paper [4]. Miller observed that a low population and high number of generations are more effective for the genetic algorithms that are used to realize these circuits [4]. This paper attempts to observe the effects of applying a new applied cloning strategy to evolve optimum circuit designs. Cloning, in effect, will keep the population constraint intact while allowing for a larger search medium.

Motivation

Many researchers have explored natural biological principles and applied them to the field of genetic algorithms [5]. Because of the limited amount of natural biological principles to apply to genetic algorithms, this paper takes into account an artificial biological principle, namely cloning, and implements a genetic algorithm based on it. This method is taken into account because it allows deeper research into the effectiveness of applying this, and other artificial biological principles to evolutionary computing.

There has also been various works in the area of evolvable hardware and FPGA design. Genetic algorithms for these particular problems have difficulty converging to optimum design solutions. By applying this new method of selection and genetic operators, a possible better optimization strategy could be unlocked.

2. EVOLVABLE HARDWARE

Background

Evolvable hardware uses evolutionary approaches to design 100% functional logic, arithmetic, and electrical circuits [3,4]. Genetic algorithms (GA) are used for synthesizing these circuits [3]. Genetic algorithms are search and optimization algorithms adapted from biological evolutionary principles [3].

Sometimes these hardware devices require speedy reconfiguration (e.g. faulty input lines or hardware used for learning) through a use of an automated process. This situation is where genetic algorithm techniques come into play, automating the process of realizing circuit designs that could potentially be realized faster than conventional human designs. Field programmable gate arrays are one such hardware device that can realize various logic circuits through evolutionary approaches.

Field Programmable Gate Arrays

Field programmable gate arrays are a subset of field programmable devices (FPD) [1]. FPGA are configurable devices consisting of an array of uncommitted circuit elements and interconnection resources [1]. The end user is able to program the FPGA to realize versatile logic configurations. FPGAs are composed of lookup tables (LUT), configurable logic blocks (CLB), and an interconnection network as seen in Figure 1 [1]. Figure 1(a) illustrates a top level example of an FPGA with CLBs and the interconnection network. Figure 1(b) illustrates the internals of a CLB. The FPGA has standard logic inputs, outputs, and a clock [1]. The lookup tables are multiplexers with the inputs tied to memory [1]. The configurable logic blocks contain the lookup tables and are connected to the interconnection network. Through the interconnection network, a CLB input can come from other CLB outputs or FPGA inputs. A CLB output can be routed to other CLB inputs, and FPGA outputs. The interconnection network is responsible for correctly routing each of the inputs and outputs to their respective locations within the FPGA. All of the cells are dictated by the use of an external clock.

FPGA Abstractions

For purpose of simplification, many of the attributes of the FPGA are removed, while retaining the main functionality of the FPGA [3]. The configurable logic blocks are removed from

the design and the lookup tables are single individual logic entities in the FPGA connected to the interconnection network. The interconnection network limitations are removed to allow any LUT or FPGA input to connect to any other LUT or FPGA output. Therefore the routing of the LUTS can accommodate many versatile configurations. The only restriction to the interconnection network is that it must be a feed-forward network, in which the output routing from one LUT, with LUT number ***l***, can only be done to another LUT with LUT number greater than ***l***, or to any FPGA output. This is implemented to prevent feedback loops [4].

The lookup table themselves will act as if there is no clock attached to the FPGA. The fixed bit-string will remain constant throughout the FPGA evaluation. The LUTs are assumed to be simple 4:1 multiplexers. The fixed bit-string is composed of four input bits, allowing any of the sixteen possible configurations. Some FPGA experiments limit the fixed bit-string to logical operators [4]. In this experiment, the fixed bit-string is not limited to allow for diverse configurations. The output bit is a single bit from the fixed bit-string chosen by the two input lines to the LUT.

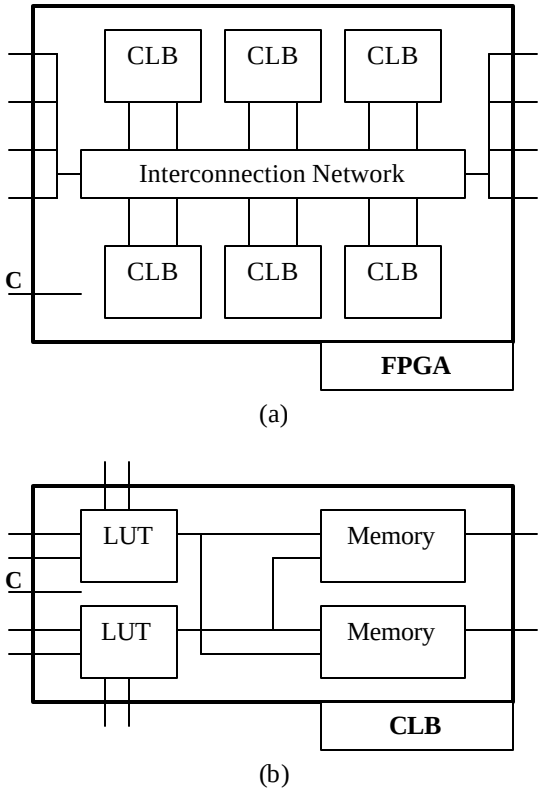

Figure 1: (a) Internal circuit diagram to the FPGA with CLBs and the Interconnection Network. (b) Diagram of the CLB subsection of an FPGA.

3. GENETIC ALGORITHM PROPERTIES

Genetic Representation

The genome in the genetic algorithm will represent the entire FPGA, by integrating the individual LUTs, their respective input routing, and the FPGA input and output routing into the encoded genome. Figure 2, illustrates the make-up of the genome with an FPGA having ***n*** LUTs and ***m*** outputs. The LUTs are composed of a binary fixed bit-string and an integer input routing scheme. Each output is represented by an integer representing the output routing. The input/output routing scheme is represented by an input/output table where each FPGA input and each LUT is represented by a unique routing number. If there are ***i*** inputs, and ***n*** LUTs, there are ***i+n*** unique routing numbers, where ***0*** to ***i-1*** are reserved for the inputs and ***i*** to ***n+i-1*** are used for the LUTs. The integer routing number is used in the representation of inputs routing into LUTs, and to the FPGA outputs.

The fixed bit-string represents the behavior of each LUT, while the inputs to the LUTs are used to choose the certain bit in the bit string to map to the output of the LUT. A few constraints are implemented for the FPGA. The routing scheme is a feed-forward scheme where any given LUT can only receive inputs from lower numbered LUTs or FPGA input, eliminating any possibility of feedback loops. Another constraint is to never allow the fixed bit string to represent a redundant logic cell, (i.e. a fixed string represented by all 0's and all 1's).

To put the genetic representation and the FPGA into perspective, the following example is presented. Given an FPGA with 2 inputs, 4 LUTs, a bit-string with a length of 4, and 1 output, a possible configuration of the FPGA can be mapped out using the genome in Figure 3. Note, the routing number is not the same as the LUT number, therefore routing 0 and 1 refer to the inputs, and routing 2 to 5 refer to LUT 1 to LUT 4 respectively. The sample genome routes the output of LUT 4 (routing number 5) to the output of the FPGA. The 2-bit adder FPGA would consist of a more complex configuration with 4 inputs, and 3 outputs.

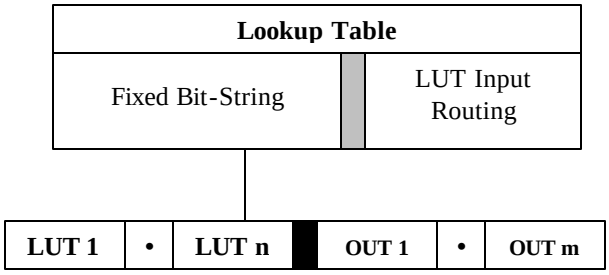

Figure 2: General Genetic Representation of an FPGA

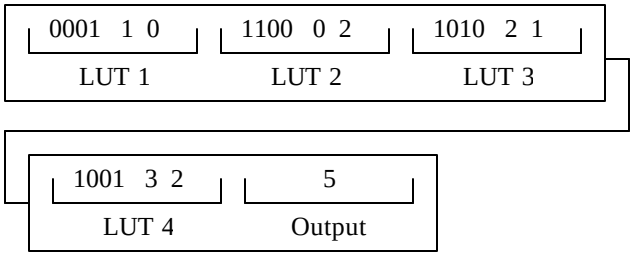

Figure 3: An Example Genetic Representation of an FPGA

Fitness Function

The fitness function of the genetic algorithm uses a logic truth table for fitness evaluation. The fitness represents the correctness of the FPGA based on the given truth table. Given binary inputs set, the FPGA is expected to produce a binary output based on the truth table. To obtain the fitness of an individual, the FPGA has to be evaluated through a computer simulation with the given inputs. The individual (genome) is given a point for every correct output in the truth table. Therefore the maximum score for the individual is the number of input/output sets in the truth table times the number of outputs for each set, which would yield the optimum individual. Therefore there is partial credit for a partial output match. The truth tables used for this paper are two-bit multipliers and two-bit adders.

Genetic Operators

There are various genetic operators that will be used in this endeavor, including mutation and one-point random crossover with elitism. Mutation will occur within the LUT, mutating the fixed bit-string, and the routing numbers. Crossover will occur between the LUTs. The two genetic operations are explained in further detail below.

Mutation: During mutation, the fixed bit string is mutated from 0 to 1 and from 1 to 0, while the routing number is mutated between input *0* and input *i+l-1* where *l* is the LUT number, and *i* is the number of inputs being mutated. The output routing will also be mutated to allow for further adaptation. This particular mutation scheme is used for the routing number to preserve the feed-forward nature of the FPGA.

One-Point Random Crossover with Elitism: During one point crossover, the LUTs are the entities subject to crossover. This is not the standard crossover method where each of the two parents is crossed over to yield two offspring. This crossover method takes two random individuals from the current population. One of the individuals could possibly be replaced with the elite individual based on the elite member crossover rate. A one-point crossover is performed on the two parents, and one of the resultant individuals is chosen at random to be inserted into the offspring population. This crossover is performed only between the LUTs, and not on the output routing portion of the genome.

Selection

The modified approach to this genetic algorithm includes applied cloning. This method is based on the artificial biological principle of cloning. Instead of individual cloning, which has a good possibly of occurrence with selection, the entire population undergoes cloning, and results in two identical population sets. Each population set undergoes the standard genetic operators and fitness evaluation in parallel. After the fitness evaluation, the two populations are compared and the population with the higher average fitness is kept, and the other population will be discarded. This method is used to maintain a low population size, while providing a larger search space on the population. Figure 4 illustrates a flow diagram of the process. The elitist selection strategy is performed on the single population instead of the cloned population. The cloned population selection goes before the elitist individual selection because it yields a better pool of individuals to choose from to make the resultant population in the next generation.

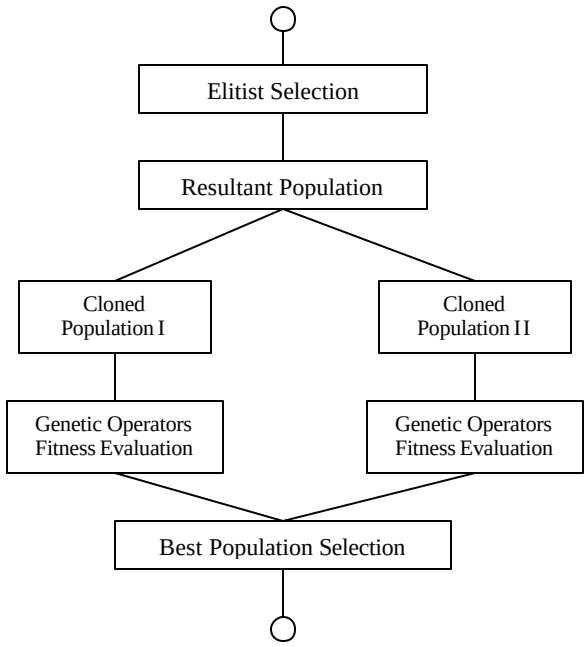

Figure 4: Block Diagram Algorithm for Applied Cloning

4. EXPERIMENTAL DES IGN

Genetic Algorithm Parameters

Population Size = 50, 100, number of generations = 10,000, number of runs 25, crossover rate = 100%, elite member crossover rate = 20%, mutation rate 5%, selection method = elitism, cloning.

Low population sizes and high generation number was chosen to reflect Miller's implementation [4]. The crossover rate was set to 100% due to the nature of the random crossover method. Each of the resultant individuals in the offspring population will have to undergo crossover and possible crossover with the elite individual. The number of runs is limited to 25 due to the processor intensive nature of the problem.

FPGA Parameters

Number of LUTS = 16, Fixed Bit-String Length = 4, Number of Inputs = 4 (2-bit adder, and 2bit multiplier), Number of Outputs = 3, 4 (3 for the 2-bit adder, with a carry bit, 4 for the 2-bit multiplier)

The FPGA Parameters are chosen arbitrarily through the observations of the various papers [2, 3, 4]. The number of LUTS implemented in the FPGA is sixteen because it allow for an adequate number of LUTs to evolve. All of the units are not necessary used for in an optimized solution, they are there to simply provide additional diversity in the population.

Design Cases

Following Miller's paper, the circuits that are used in the experimental design are the two-bit adder, and the two-bit multiplier [4]. Truth tables will be used in the circuit design. Each of the circuits will undergo evaluation with a standard genetic algorithm, a clone implemented genetic algorithm, and a

standard genetic algorithm with double the population. Since cloning is essentially doubling the population, the success of cloning implementation is also compared to that of a standard GA with double the population. There are a total of six evaluations, three for each circuit.

5. RESULTS AND DISCUSSION

Results

The results of the six evaluations are discussed below. One interesting phenomena observed is the deceptive nature of the problem. The population tends to increase in fitness significantly in the first few generations then stabilizes. The large subsequent number of generations is used to search for the optimum solution. For the adder, the optimum individual exhibits a maximum fitness of 48 (16 truth table lines * 3 outputs), while the multiplier has a maximum fitness of 64. The sections below will compare the cloned population to the standard population and the cloned population to the doubled population.

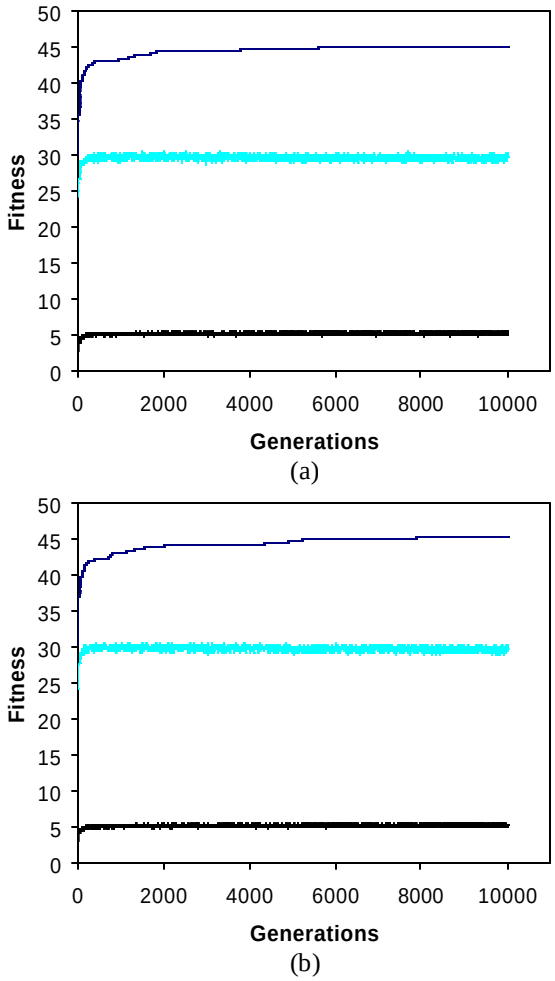

Figure 5: Graph of the Best Fitness (Top Line), Average Fitness (Middle Line), and Standard Deviation (Bottom Line) for the Two-Bit Adder Circuit with the (a) *Standard* Implementation and (b) *Cloned* Implementation.

Cloning vs. Standard Population

This comparison is strictly measured based on the population size. The graphs in Figures 5 (a) and (b) represent the growth and behavior of the standard GA and the cloned GA respectively for the two-bit adder implementation. The graphs for the two-bit multiplier are similar to the above graphs, and are not shown. The results at generation 10000 are listed in Table 1 for both the adder and the multiplier.

Table 1: Table of the Average and Best Fitness at Gen. 10000 for the Standard and Cloned Population Runs.

Two-Bit Adder	Average	Best	Std. Dev.
Standard (1)	29.52	45.04	5.06
Cloned (2)	29.41	45.20	5.17
Two-Bit Multiplier	Average	Best	Std. Dev.
Standard (0)	42.54	59.96	7.63
Cloned (0)	42.78	59.72	7.41

At first glance, both of the populations performed equally well. There is no significant different in the numbers. The standard population realized one optimum individual and the cloned population two. No optimum multiplier circuit was discovered between the two implementations. The only major difference between the two populations is the computational power involved in reaching a solution. The applied cloning method did not increase the efficiency of the GA.

Another interesting observation was the average fitness does not exactly converge on a single value, yet, it converges on a general fitness region. The average fitness remains within a constant band after the growth period. After that initial growth period, it seems the GA populations are not making an improvement and have stabilized. To reach higher fitnesses, it seems the GA has to essentially get lucky and stumble upon a higher fit individual.

Cloning vs. Doubled Population

This comparison is measured based on computational time. Cloning essentially represents a doubled population and is therefore compared to that of a GA with a doubled population. Figure 6 represents the behavior of the doubled population for the two-bit adder implementation. The results at generation 10000 are listed in Table 2 for both the adder and the multiplier.

Again, the doubled population exhibited no significant difference compared to the cloned population. The doubled population was able to find a much larger number of optimum solutions to the problem. This is perhaps due to the large range of individuals in the population. This gives the population more chances to get a lucky individual has increased its fitness.

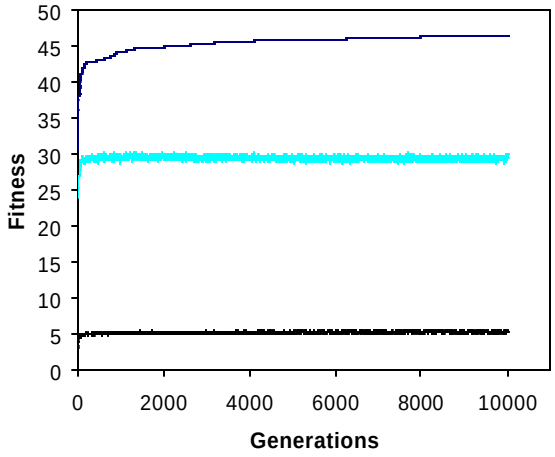

Figure 6: Graph of the Best Fitness (Top Line), Average Fitness (Middle Line), and Standard Deviation (Bottom Line) for the Two-Bit Adder Circuit with the *Doubled Population* Implementation.

Table 2: Table of the Average and Best Fitness at Gen. 10000 for the Doubled and Cloned Population Runs.

Two-Bit Adder	Average	Best	Std. Dev.
Double (4)	29.30	46.28	5.32
Cloned (2)	29.41	45.20	5.17
Two-Bit Multiplier	Average	Best	Std. Dev.
Double (4)	41.49	60.96	9.60
Cloned (0)	42.78	59.72	7.41

Discussion

Due to the deceptive nature of the problem, the genetic algorithm is split into two sections, the growth section and the stabilize section. The initial 200 generations is the growth section used to mainly reach a convergent fitness (where the growth of the individual tends to stabilize), while the remaining 9800 generations is the stabilize section used to attempt to reach the optimum fitness. The nature of the problem makes it difficult to compare the effect of cloning on the population.

Cloning divides the search space into two different, but possibly very similar search areas. The better fitnessed population is chosen over the other one in the case of cloning. The problem has many cases where the truly better landscape section is discarded. Cloning has increased chances of choosing a population that would lead the GA close to the optimum fitness during the run, but due to the nature of this problem and the fact the GA has to essential get lucky and stumbles upon a better fit individual, the GA does not perform exactly in this manner. Instead the cloned population performs equally to that of the standard GA. The cloning process does not have a visible impact on the performance of the GA. Compared to the computational power required by the cloned population, a doubled population would be more suitable in this implementation.

6. CONCLUSION

The applied cloning strategy is a new selection method for genetic algorithms discussed in this paper. The goals of this paper were to introduce this new method to the domain, and show some initial testing data with the new method. The first testing of applied cloning does not show significant increases on GA performance. It does however show a minor increase in convergence toward the optimum fitness. Cloning has its merits with the improvement of genetic algorithms in this problem domain. This study unlocked the difficult nature of the problem domain, by illustrating how the problem reaches the many local optimum points in the fitness landscape.

Because the problem is not an "easy" problem to solve, the effects of cloning are not as noticeable as had earlier hoped. The benefits of cloning appear to be not worth the computational cost. This problem already requires a large amount of computational power to execute, and instead of applying a cloning method, an increase in population or generation could in effect be more beneficial to this type of problem. There are other problems that could potentially benefit from the applied cloning strategy. These tests are only the initial tests performed with this strategy; concrete conclusions cannot be made about this method until further tests are conducted.

Contributions

There are two contributions made by this work in the fields of genetic algorithms and evolvable hardware. The research put forth considers a new approach to genetic algorithms and could possibly lead to more effective means of optimizations. This paper has the potential to introduce new approaches in the evolvable hardware field, and lend itself to new perspectives on different genetic algorithm implementation approaches to common evolvable hardware problems.

7. FUTURE WORK

Because this strategy is new, it should be tested with standard GA problems, for instance the max ones or the royal road problem. By using a less complicated problem, and a more even fitness landscape, perhaps the true potential of applied clone can be discovered.

Cloning yields two identical populations and the genetic operators to each of the cloned populations yielded different unique populations. Yet, resultant populations remained fairly similar through the use of conservative genetic operators implemented in this paper. Future work could entail the addition of more radical genetic operators to the cloned population, adding more diversity to the separate populations, and allowing the genetic algorithm to explore more regions of the search space. Another method to lessen the disruptive effects of the genetic operators is to use the conservative set of genetic operators on one cloned population and the radical set on the other.

8. REFERENCES

[1] Brown, S., and Rose, J., "FPGA and CPLD Architectures: A Tutorial", *IEEE Design & Test of Computers*, pp. 42-57, 1996.

[2] Lohn, J., Larchev, G., and DeMara, R., "A Genetic Representation for Evolutionary Fault Recovery in Virtex FPGA's", *The 5th International Conference on Evolvable Systems: From Biology to Hardware*, 2003.

[3] Keymeulen, D., Zebulum, R.S., Jin, Y., and Stoica, A., "Fault-Tolerant Evolvable Hardware Using Field-Programmable Transistor Arrays", *IEEE Transactions on Reliability*, Vol. 49, No. 3, pp. 305-316, 2000.

[4] Miller, J.F., Thomson, P., Fogarty, T., "Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study", *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, 1998.

[5] Grefenstette, J. J., "Optimization of Control Parameters for Genetic Algorithms", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol 16, No. 1, pp. 122-128, 1986.